Attorney Docket No.: 010327-008600US
Client Reference No.: Log #34

PATENT APPLICATION

METHOD AND APPARATUS FOR DETECTION OF HOSTILE SOFTWARE

Inventor: Michael D. Brent,
A Citizen of Canada
36276 Worthing Drive
Newark, CA 94560

Assignee: NETWORK EQUIPMENT TECHNOLOGIES, INC.
6900 Paseo Padre Parkway
Fremont, CA 94555

Entity: Large

TOWNSEND and TOWNSEND and CREW LLP
Two Embarcadero Center, Eighth Floor
San Francisco, California 94111-3834
Tel: 650-326-2400

METHOD AND APPARATUS FOR DETECTION OF HOSTILE SOFTWARE

BACKGROUND OF THE INVENTION

[0001] As computer systems become more and more interconnected with the expansive use of computer networks, the threat of infection by hostile software transported over such computer networks has steadily increased. Here, hostile software may include any software that could affect a computer system in an undesirable or otherwise unwanted manner. Traditional anti-virus techniques have curtailed the spread of hostile software to a certain extent. However, such techniques typically rely on some type of inspection of a computer system for known hostile software signatures. Thus, such an approach is often incapable of handling hostile software for which a signature has not been positively identified, such as in the case of hostile software that has not yet been created. Even for existing hostile software, delays associated with identification of the hostile software's signature may lead to additional damage and further opportunity for the hostile software to re-spawn.

[0002] Furthermore, as hostile software become increasingly sophisticated, the task of identifying such signatures may become significantly more difficult and time-consuming, or worse, may become impossible in some instances. For example, hostile software may mutate such that a previously identified signature for the original hostile software may no longer be useful in identifying the mutated version of the hostile software. Thus, there exists an urgent need for improved techniques to detect and respond to new and more sophisticated hostile software in the face of the ever higher usage of network-enabled computer systems.

BRIEF SUMMARY OF THE INVENTION

[0003] The present invention relates to methods and apparatuses for detecting hostile software in a computer system involving storing a representation of configuration data associated with an operating system for the computer system obtained at a first time, comparing the stored representation of the configuration data obtained at the first time with a representation of the configuration data associated with the operating system for

the computer system obtained at a second time, and if deviation is detected between the stored representation of the configuration data obtained at the first time and the representation of the configuration data obtained at the second time, automatically performing at least one remedial measure in response to the deviation detected.

[0004] In one embodiment of the invention, the configuration data relates to identification of executable code installed in the computer system. The configuration data may also relate to identification of a command line for invoking executable code associated with a particular file extension. The configuration data may be obtained from a registry key in a registry maintained by the operating system. The configuration data may also be obtained from a file stored in the computer system. The stored representation of configuration data is encoded prior to being stored.

[0005] The at least one remedial measure may comprise determining a storage location associated with suspected executable code in the computer system, determining whether suspected executable code is currently executing, terminating execution of the suspected executable code without notification, moving suspected executable code to a specified storage location for later evaluation, and/or altering configuration data associated with the operating system to reflect the stored representation of the configuration data.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] Fig. 1 is a block diagram of a computer system, connected to a network, capable of implementing techniques for detecting and reacting to hostile software in accordance with one embodiment of the present invention;

[0007] Fig. 2 illustrates a user interface representation of a registry having a tree-like structure and containing keys having configuration data utilized in various embodiments of the present invention;

[0008] Fig. 3 is a flow chart outlining basic steps for storing configuration data in accordance with one embodiment of the present invention;

[0009] Fig. 4 is a flow chart outlining basic steps for detecting hostile software based on configuration data and comparing stored configuration data to that of the current

system, detecting a deviation, and performing a reaction or recovery task in accordance with one embodiment of the present invention;

[0010] Fig. 5 is a more detailed flow chart outlining steps in one example of detecting hostile software in the context of a Windows-based operating system in accordance with one embodiment of the present invention;

[0011] Fig. 6 is a more detailed flow chart outlining steps in one example of detecting hostile software in the context of a Linux-based operating system in accordance with one embodiment of the present invention; and

[0012] Fig. 7 is a more detailed flow chart outlining steps for reacting to detection of hostile software in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0013] Fig. 1 is a block diagram of a computer system 100, connected to a network 102, capable of implementing techniques for detecting and reacting to hostile software in accordance with one embodiment of the present invention. Computer system 100 represents any computing devices that may be subject to a threat of hostile software.

[0014] Computer system 100 contains an operating system 104 for carrying out basic functions such as file management within computer system 100. As shown in this figure, operating system 104 contains primary applications 106, registry/start up file 108, and detection system 110. Although not shown in Fig. 1, the computer system 100 typically includes a processing unit for carrying out machine instructions and a storage system for storing data. Primary applications 106 are applications that may be executed in computer system 100. While shown in this figure as being contained inside operating system 104, primary applications 106 may actually be stored at a physical location distinct from that of operating system 104. Registry/start up file 108 contains configuration data such as a list of applications to execute when the operating system 104 is initiated and other information relevant to the configuration of operating system 104. Here, registry/start up file 108 may comprise a software entity referred to as a registry that contains different configuration data. Alternatively or in addition, registry/start up file 108 may comprise one or more files for storing configuration data. Detection system 110 contains code for implementing detection of and reaction to hostile software in accordance with this embodiment of the invention. Thus, detection

system 110 may comprise any executable code, such as a script file, application program, system driver, and/or hardware, such as those used to monitor the computer system 100.

[0015] Here, computer system 100 may be a personal computer, a server, or any other device utilizing an operating system. In particular, computer system 100 may comprise or be part of a closed system that does not receive regular user interaction in its normal operations. Some examples of closed systems include routers, embedded computer systems such as automobile computers and computers used in manufacturing, telephony switches, network access servers such as those used in connection with dial-up connection services and Internet service providers. Such closed systems are typically installed, configured, and then left alone unless a problem occurs. Advantages provided by the present invention may prove especially beneficial in allowing such closed systems to perform self-monitoring and attempt recovery should an attack by hostile software occur.

[0016] Hostile software may be introduced to computer system 100 via network 102. As shown in Fig. 1, network 102 is also connected to one or more other computer systems 112 and 114. Certainly, network 102 may be connected to other computers and other networks. Network 102 may be a private network or a public network. Network 102 may be connected to or be part of a local area network or wide area network. Indeed, network 102 may be connected to or be part of what is commonly referred to as the Internet. As more and more devices such as computer system 100 become network-enabled, there is an increasing risk of attack by hostile software introduced via a network connection.

[0017] Fig. 2. illustrates a user interface representation of a registry having a tree-like structure and containing keys having configuration data utilized in various embodiments of the present invention. Many types of operating systems may organize configuration data useful for hostile software detection in a software entity referred to as a registry. The registry may be viewed as a database of settings maintained for the operating system. It is often the case that information in the registry is backed up in memory when the operating system is running, so that access to the registry may create very little if any disruption to the active operating system. As shown in Fig. 2, a registry typically has a tree-like structure reminiscent of the organization of files in

directories or folders. A number of registry keys in the registry are visible in Fig. 2, including:

HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Run

HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/RunOnce

HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/RunOnceEx

[0018] Examples of operating systems that organize configuration data in a registry include Microsoft Windows-based operating systems such as Windows NT, Windows NT Server, Windows 95, Windows 98, Windows 2000, Windows 2000 Professional, Windows XP Home, Windows XP Professional, and Windows XP Embedded.

[0019] Configuration data such as that shown in the registry illustrated in Fig. 2 may contain information relating to executable code that may be executed in the computer system. Here, executable code broadly refers to any portion of code capable of being executed in a computer system, such as a script file, application program, system driver, and others. Thus, the configuration data may include, for example, a list of application programs that are to be launched when the operating system is initiated.

[0020] Fig. 3. is a flow chart 300 outlining basic steps for storing configuration data in accordance with one embodiment of the present invention. In a step 302, an operating system such as operating system 104 in computer system 100 is started. In a step 304, the startup system of the operating system is examined to obtain relevant configuration data that may be useful for hostile software detection. The configuration data may be obtained from one or more keys in a registry such as shown in Fig. 2. Generally speaking, the configuration data that is obtained may or may not be an exact copy of the configuration data maintained by the operating system. For example, the configuration data obtained may be a subset or summary of the configuration data maintained by the operating system. Thus, the data obtained may also be referred to here as a representation of the configuration data. In a step 306, the configuration data obtained from the examination is encoded. Here, encoding broadly refers to the transformation of data into a different format. Thus, encoding may include encryption, cryptography, obfuscation, and/or other techniques. Next, in a step 308, the result of

the encoding is stored. Storing the configuration data in an encoded format make it less vulnerable to manipulation by hostile software, which may attempt to add itself as a valid entry in the stored configuration data. At step 310, the process for storing configuration data according to the present embodiment is complete.

[0021] Fig. 4 is a flow chart 400 outlining basic steps for detecting hostile software based on configuration data and comparing stored configuration data to that of the current system, detecting a deviation, and performing a reaction or recovery task in accordance with one embodiment of the present invention. In a step 402, the detection process is started. In a step 404, the process enters a sleep state. Whenever detection of hostile software is desired, the process may exit the sleep state to perform steps necessary for such detection. Accordingly, the process may exit the sleep state on a regular basis, such as once every 15 minutes, to perform detection tasks. Other schedules or arrangements for determining when to exit the sleep state are certainly possible. Upon exit from the sleep state, in a step 406, the stored configuration data is compared with current configuration data obtained from the active operating system. Again, the current configuration data obtained from the active operating system may or may not be an exact copy of the configuration data maintained by the operating system. Thus, the data obtained may also be referred to here as a representation of the configuration data. If the comparison performed in step 406 results in the detection of a deviation between the stored configuration data and the current configuration data, the process moves to a step 410. Else, the process returns to step 404. In step 410, at least one reaction or recovery task is performed for each deviation detected. Upon completion of step 410, the process returns to step 404.

[0022] Fig. 5 is a more detailed flow chart 500 outlining steps in one example of detecting hostile software in the context of a Windows-based operating system in accordance with one embodiment of the present invention. The steps shown in Fig. 5 may correspond to or replace step 406 in Fig. 4. In a step 502, a particular process for detecting hostile software in a Windows-based operating system is initiated. In a step 504, a registry such as the registry illustrated in Fig. 2 is opened. In a step 506, various registry keys such as Run and RunServices keys under ROOT and all USERS are scanned for any deviation from stored configuration data.

[0023] For example, the first registry key used for program startup in a Windows-based operating system may be HKEY_LOCAL_MACHINE/Software/Microsoft/Windows/RunServices. This key is typically scanned when the system starts, and programs listed under it are started. This startup key may be monitored for any additions or other changes. Certain systems may not have this key if there are no programs to be launched at that time. As another example, should a user log in to the operating system, two other keys may be scanned to run programs -- one under HKEY_LOCAL_MACHINE, and one under HKEY_CURRENT_USER (which is an alias for the user who is logged in, under HKEY_USERS). In both cases, the relevant key is identified as Software/Microsoft/Windows/Run. All entries under each of these keys are started with the shell of the operating system. These startup keys may also be monitored for any additions or other changes. Once again, if there are no programs to be launched at that time, the keys may not exist. As yet another example, a portion of the registry referred to as the services branch, located at HKEY_LOCAL_MACHINES/System/CurrentControlSet/Service may also be monitored. By detecting change in configuration data, such as addition of new keys, to this portion of the registry, hostile software that attempts to bypass the shell startup sequence of the operating system may also be detected. Also, if file system monitoring on a limited scale is deemed acceptable, then scanning the "Startup" folder of all users, located under the Documents and Settings folder at <user name>\Start Menu\Startup (including under the user 'All Users') is another location from which the shell may load programs. Changes to configuration such as addition of new shortcuts or executables appearing in this folder may also indicate existence of hostile software.

[0024] Next, in step 508, executable keys associated with execution of software programs having a particular file extension are scanned. For example one section of the registry is referred to as HKEY_CLASSES_ROOT, which is an alias for HKEY_LOCAL_MACHINE/Software/Classes. This key contains information for files by extension, including identification of the command line to execute when a file is opened. It may be possible for hostile software to insert itself into the execution path for common extensions, thus ensuring it is executed any time the shell opens a file. The main executable file types in Windows-based operating systems may include EXE, COM, SCR and PIF. Thus, under HKEY_CLASSES_ROOT, the command line may

be monitored by watching <ext>file/Shell/Open/Command, on it's default value. For the command types listed above, the value should be:

"%1" %*

[0025] This value indicates that the program requested (%1) should be executed, and it's arguments are all other arguments passed (%*). If this string is replaced, then any file of that type which is started by the shell will use the new command line. For example, a hypothetical virus with a program name of "VIRUS.EXE" could replace the above string under the 'exefile/Shell/Open/Command' with the following:

VIRUS.EXE "%1" %*

[0026] This would cause every executable file ending in .EXE to start the Virus process first (which could then, in turn, start the actual program in order to hide the fact that it was triggered). Thus, step 508 may scans the proper executable key for a correct pattern, such as the default value "%1"%* described above and conclude that hostile software has been detected if deviation from the correct pattern is found.

[0027] In a step 510, various registry keys such as installed services are scanned for any deviation from the stored configuration data. Step 510 may be similar to step 506 in operation. In fact, steps 506 and 510 may be performed in succession or combined in some manner, depending on implementation. Next, in a step 512, the registry is closed. Finally, in a step 514, this process for detecting hostile software is completed.

[0028] Although not illustrated in Fig. 5, other techniques for detecting hostile software in a Windows-based operating system may be alternatively or additionally implemented. First, removal of configuration information, or failure to remove configuration where such removal is expected, may also indicate existence of hostile software. For example, the following keys are primarily used for upgrades in Windows-based operating system, and therefore may not be found on all machines:

HKEY_LOCAL_MACHINE/Software/Microsoft/Windows/RunServicesOnce

HKEY_LOCAL_MACHINE/Software/Microsoft/Windows/RunOnce

HKEY_CURRENT_USER/Software/Microsoft/Windows/RunOnce

[0029] Under normal operations, entries under these key may exist when they are executed, but are then removed from the registry by the operating system. Hostile

software may attempt to escape detection by storing its startup commands under such a key, waiting for the operating system to remove entries under the key, then restore its data under the key. Thus, in one embodiment of the invention, hostile software detection may involve sensing that one or more keys/entries exist and later verifying that they are removed after a restart of the operating system. In the practical application, all HKEY_USERS keys may be tested, not just the CURRENT user.

[0030] Second, files separate from the registry may be useful in detection of hostile software in the context of a Windows-based operating system. Even though some of these files may exist and not be used in certain Windows-based operating systems such as Windows 2000 or XP, monitoring of these files may nevertheless assist in the detection of hostile software. For example, the files WIN.INI and SYSTEM.INI in the Windows installation folder, and WINSTART.BAT in the root of the file system may be monitored because hostile software may attempt to modify these files to cause themselves to be loaded under different Windows-based operating systems such as Windows 95, 98 and ME. Thus, it may be the case that the hostile software would not have caused damage by modifying these files, but such file modifications could be utilized to achieve detection of the hostile software. For example, in the WIN.INI file, the keys LOAD= and RUN= may be monitored for additions, which may be complete paths, or program files in the search path. In the SYSTEM.INI file, the SHELL= line may be checked for modifications. Further, the mere existence of the WINSTART.BAT file may indicate possible hostile software activity, since this particular file was intended for use with a much earlier Windows-based operating system, Windows 3.1, although it was still supported under Windows 95 and 98.

[0031] Third, the shell configuration may also be monitored to assist in the detection of hostile software. The shell configuration may be specified in older versions of Windows in the text-based SYSTEM.INI file, but is transparently remapped to the registry in NT-based versions of Windows. It resides at HKEY_LOCAL_MACHINE/Software/Microsoft/Windows NT/CurrentVersion/Settings, and it should read "Explorer.exe." Any changes may indicate existence of hostile software. Thus, different steps in Fig. 5 may be added, deleted, combined, or otherwise altered to accomplish hostile software detection in different ways in accordance with various embodiments of the present invention.

[0032] Fig. 6 is a more detailed flow chart 600 outlining steps in one example of detecting hostile software in the context of a Linux-based operating system in accordance with one embodiment of the present invention. The steps shown in Fig. 6 may correspond to or replace step 406 in Fig. 4. While this embodiment is specific to Linux-based operating systems, the invention is not necessarily so limited and may be utilized in other Unix-derived operating systems.

[0033] Like other operating systems, a Linux-based operating system typically includes means for identifying software that is to be started with the system. Accordingly, as a Linux-based operating system is initiated, it may pass through a series of what are referred to as runlevels. Just as an example, the runlevels defined for the popular Red Hat Linux are as follows:

0 — Halt

1 — Single-user mode

2 — Not used (user-definable)

3 — Full multi-user mode

4 — Not used (user-definable)

5 — Full multi-user mode (with an X-based login screen)

6 — Reboot

[0034] As the operating system enters each run level, a pre-defined script is executed to prepare that level for use. These scripts may, in turn, launch other scripts, therefore detection under this operating system may be required to be aware of all scripts.

[0035] Thus, according the current embodiment of the invention, in a step 602, a particular process for detecting hostile software in a Linux-based operating system is initiated. In a step 604, a startup script at a first runlevel is opened. In a step 606, the script that has been opened is examined for changes. Here, changes may be detected by comparing the script to a stored version of the script. The stored version of the scripts may have been previously stored as illustrated in Fig. 2. The script may launch or otherwise reference other scripts external to itself. Such external scripts may also be examined for changes in a similar manner. Next, in a step 608, any included folders associated with the script may also be examined for changes. Changes may include any

deviation such as modification, addition, or removal of data, such as new or altered files. In a step 610, a determination is made whether all scripts in the current runlevel has been examined. If so, the process moves to a step 614, where the process reaches a conclusion. If not, the process moves to a step 612 to open a script at the next runlevel. Following step 612, the process moves back to step 606 for examination of script(s). Thus, in this particular embodiment, an exhaustive examination is conducted of all scripts at all runlevels of the Linux-based operating system. However, the invention is not so limited and may certainly examine only a portion of the existing scripts.

[0036] Although not illustrated in Fig. 6, the process may also utilize a time-scheduler system, known on most Linux-based operating systems as cron, to schedule various steps such as those shown in Fig. 6 to be performed at pre-determined time(s).

[0037] Fig. 7 is a more detailed flow chart 700 outlining steps for reacting to detection of hostile software in accordance with one embodiment of the present invention. The steps shown in Fig. 7 may correspond to or replace step 410 in Fig. 4. A step 702 indicates the beginning of these steps. At a step 704, an attempt is made to identify suspected executable code and a corresponding file path, based on the change(s) detected between the current configuration data and the stored configuration data. These change(s) may be detected by performing steps such as those illustrated in Figs. 4, 5, and 6. The suspected executable code may be stored in a file, and the file path may indicate the location of such a file in the computer system. For example, the current configuration data may list an application program, as a particular file located at a specified file path, that is not listed in the stored configuration data. Such an application program may be the suspected executable code identified in step 704. Next, in a step 706, if suspected executable code is identified in step 704, a step 708 is performed. Otherwise, a step 720 is performed. Here, it may be possible that suspected executable code is not identified in step 704, but a change is nevertheless detected between the current configuration data and the stored configuration data. This could indicate that that hostile software may be present to have caused the deviation between the stored configuration data and the current configuration data, but no executable code corresponding to the hostile software can be positively identified.

[0038] In step 708, a process list maintained by the operating system may be scanned to determine whether the suspected executable code is currently executing in the

computer system. Next, in a step 706, if the suspected executable code is executing, a step 712 is performed. Else, a step 714 is performed. In step 712, execution of the suspected executable code is terminated. In one embodiment of the invention, as the suspected executable code is being executed, it does not receive notification prior to being terminated. For instance, executable code, such as an application program, may typically be terminated in its execution by a message from a user or an operating system, instructing it to exit. This gives the program a chance to close open resources, save settings, or complete other tasks prior to termination. When the executable code is suspected of being hostile software, it may be useful to terminate execution of that code without providing such notification. This may reduce the opportunity for the suspected executable code to attempt or recover itself or perform some type of counter measure. In step 714, storage space such as a hard disk in the computer system is checked in an attempt to locate the executable code where it is stored, such as in a file. In a step 716, if such a file is located, a step 718 is performed. Else, step 720 is performed. In step 718, the file is moved to another location for later evaluation. Such a location may be referred to as a "quarantine" folder or directory. In step 720, the system is reset to match the stored state, by altering the current configuration data associated with the operating system to reflect the stored configuration data. Next, in a step 722, an operator may be notified. Finally, a step 724 indicates the conclusion of these steps.

[0039] While the present invention has been described in terms of specific embodiments, it should be apparent to those skilled in the art that the scope of the present invention is not limited to the described specific embodiments. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense. It will, however, be evident that additions, subtractions, substitutions, and other modifications may be made without departing from the broader spirit and scope of the invention as set forth in the claims.